Exhibit 1

EXHIBIT 1

CYWEE GROUP LTD,
vs.
HUAWEI DEVICE CO. LTD.,
HUAWEI DEVICE (DONGGUAN) CO. LTD., AND
HUAWEI DEVICE USA, INC.

UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EXEMPLARY CLAIM CHART

U.S. PATENT NO. 8,441,438 – Huawei Honor 8 Infringement Contentions

These contentions are disclosed to only provide notice of Plaintiff's theories of infringement. These contentions do not constitute proof nor do they marshal Plaintiff's evidence of infringement to be presented during trial.

Claim 1

Claim 1, with claim constructions, is recited below (text in brackets [] reflects the Court's claim construction or the parties' agreed claim construction in *CyWee Group, Ltd. v. Apple Inc.*, No. 3:13-cv-01853-HSG). Construed terms and constructions are underlined.

1. A three-dimensional (3D) pointing device subject to movements and rotations in dynamic environments, comprising:

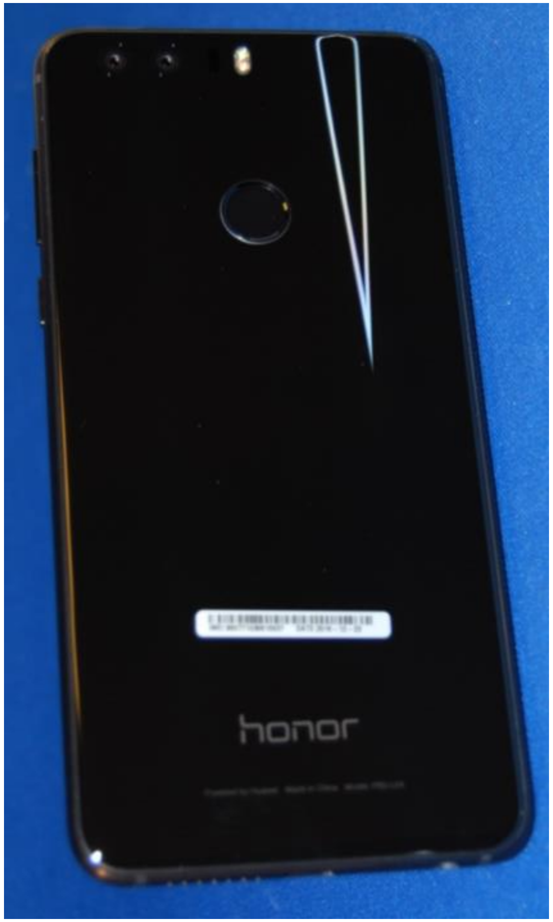
a housing associated with said movements and rotations of the 3D pointing device in a spatial pointer reference frame;

a printed circuit board (PCB) enclosed by the housing;

a six-axis motion sensor module attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame, an accelerometer for detecting and generating a second signal set comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

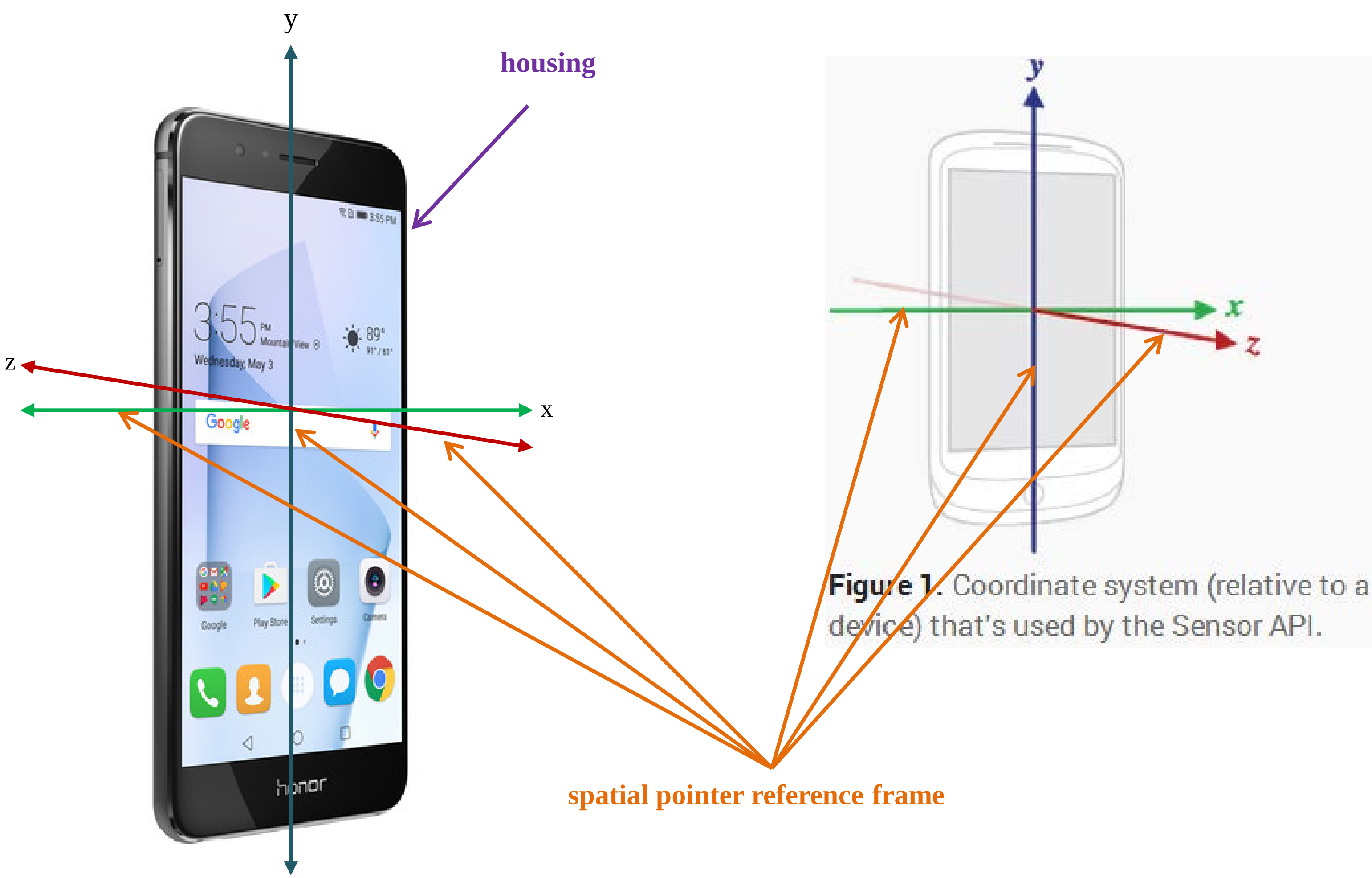
a processing and transmitting module, comprising a data transmitting unit electrically connected to the six-axis motion sensor module for transmitting said first and second signal sets thereof and a computing processor for <u>receiving and calculating said first and second signal sets from the data transmitting unit [Court's construction: no construction necessary]</u>, communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by <u>utilizing a comparison to compare the first signal set with the second signal set [Court's construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set]</u> whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments, wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an updated state based on a previous state associated with said first signal set and a measured state associated with said second signal set; wherein the measured state includes a measurement of said second signal set and a predicted measurement obtained based on the first signal set without using any derivatives of the first signal set.

Claim 1
A three-dimensional (3D) pointing device subject to movements and rotations in dynamic environments, comprising:



Huawei Honor 8

Claim 1
a **housing** associated with said movements and rotations of the 3D pointing device in a **spatial pointer reference frame**;



Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 1
a **printed circuit board (PCB)** enclosed by the **housing**;

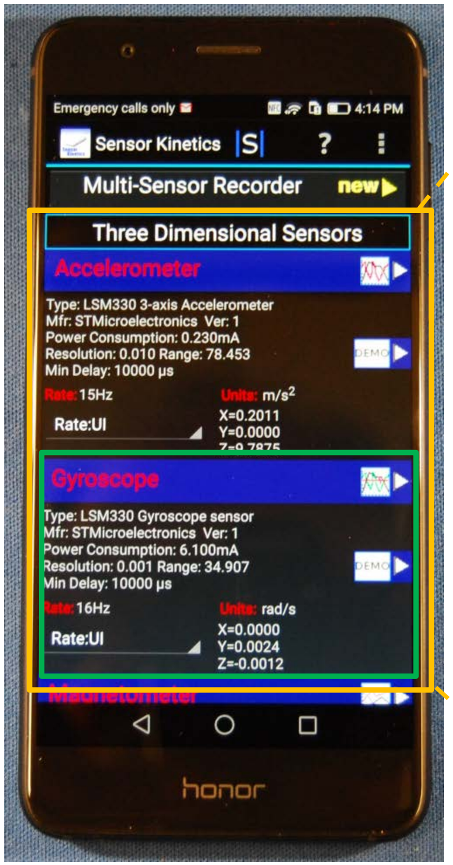
printed circuit board (PCB)

housing

Claim 1

a **six-axis motion sensor module** attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

The **six-axis motion sensor module** includes an accelerometer and gyroscope combo. The **rotation sensor** is the Gyroscope included in the **six-axis motion sensor module**.



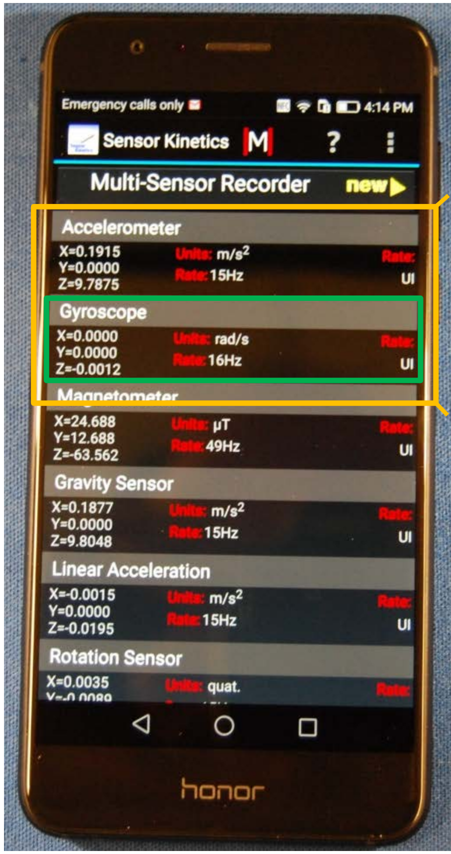
FEATURES	Sensors	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass

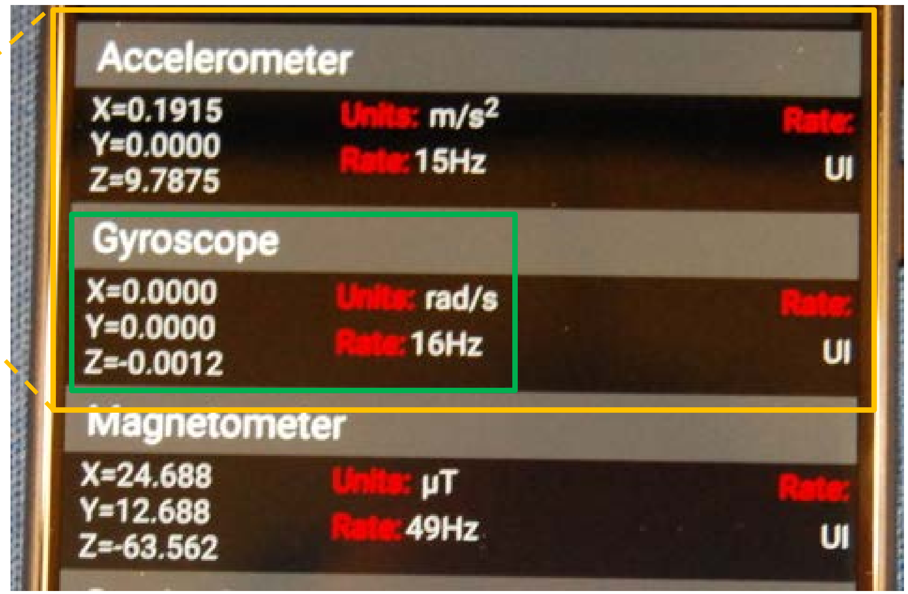
Source: http://www.gsmarena.com/huawei_honor_8-8195.php

Claim 1

a **six-axis motion sensor module** attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

The **six-axis motion sensor module** includes an accelerometer and gyroscope combo. The **rotation sensor** is the Gyroscope included in the **six-axis motion sensor module**.





FEATURES	Sensors	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass

Source: http://www.gsmarena.com/huawei_honor_8-8195.php

Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**,

The **six-axis motion sensor module** includes the accelerometer and gyroscope. The **rotation sensor** is a gyroscope.
The **first signal set** includes the sensor event values of TYPE_GYROSCOPE.

Gyroscope

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_GYROSCOPE)` *returns a non-wake-up sensor*

A gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes.

Rotation is positive in the counterclockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the aerospace definition of roll.

The measurement is reported in the x, y and z fields of `sensors_event_t.gyro` and all values are in radians per second (rad/s).

Source: https://source.android.com/devices/sensors/sensor-types#gyroscope

`Sensor.TYPE_GYROSCOPE`:

All values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis. The coordinate system is the same as is used for the acceleration sensor. Rotation is positive in the counter-clockwise direction. That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. Note that this is the standard mathematical definition of positive rotation and does not agree with the definition of roll given earlier.

- values[0]: Angular speed around the x-axis
- values[1]: Angular speed around the y-axis
- values[2]: Angular speed around the z-axis

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 1
a six-axis motion sensor module attached to the PCB, comprising a **rotation sensor** for detecting and generating a **first signal set** comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame,

Variable `w`, used by the `handleGyro()` function in the fusion.cpp file, represents gyroscope data or a **first signal set**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

a six-axis motion sensor module attached to the PCB, comprising a rotation sensor for detecting and generating a first signal set comprising angular velocities ω_x, ω_y, ω_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**,

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

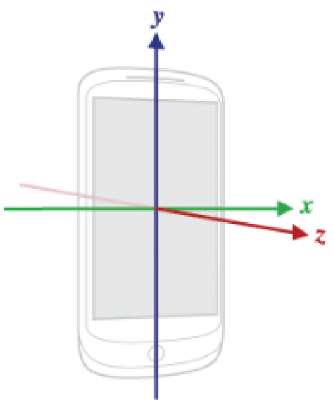


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

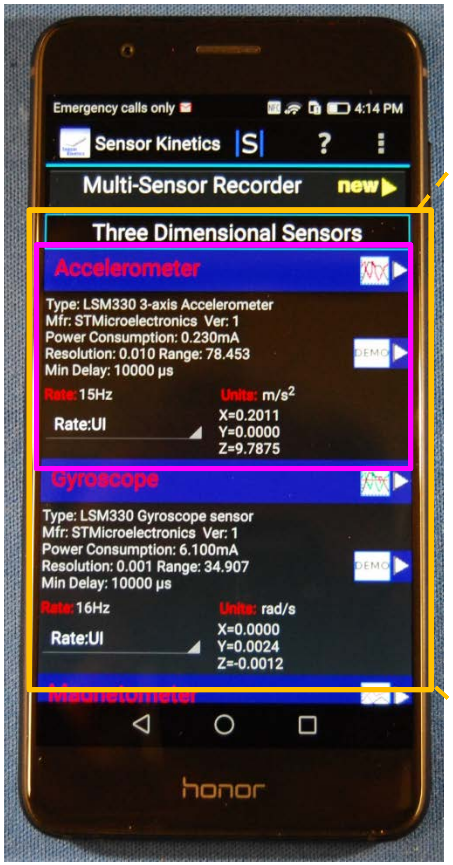
The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 1
a **six-axis motion sensor module** attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and

The **six-axis motion sensor module** is an **accelerometer** and gyroscope combo.



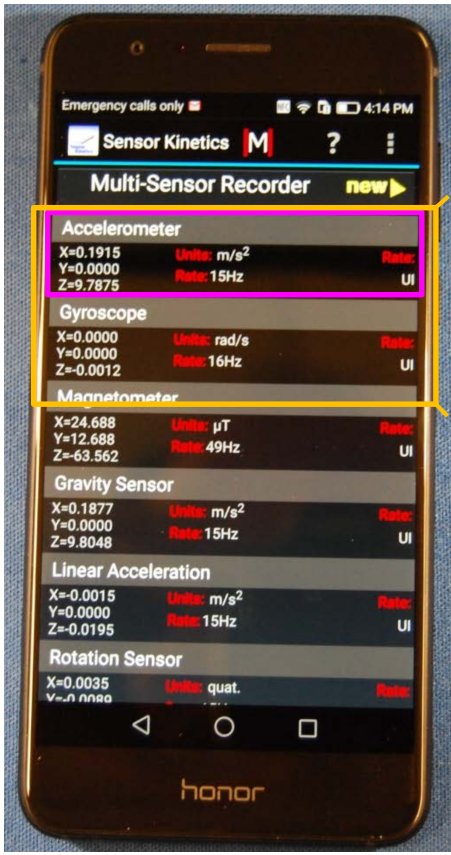
FEATURES	Sensors	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass

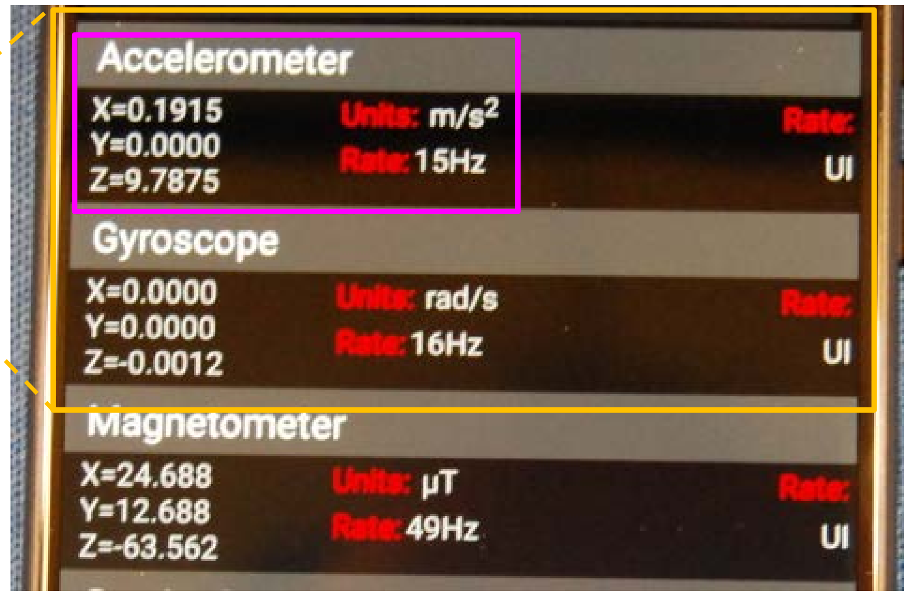
Source: http://www.gsmarena.com/huawei_honor_8-8195.php

Claim 1

a six-axis motion sensor module attached to the PCB, comprising…an accelerometer for detecting and generating a second signal set comprising axial accelerations Ax, Ay, Az associated with said movements and rotations of the 3D pointing device in the spatial reference frame; and

The six-axis motion sensor module is an accelerometer and gyroscope combo.





FEATURES	Sensors	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass

Source: http://www.gsmarena.com/huawei_honor_8-8195.php

Claim 1

a six-axis motion sensor module attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

The six-axis motion sensor module also includes an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations. The **second signal set** includes the sensor event values of TYPE_ACCELEROMETER.

Accelerometer

Reporting-mode: *Continuous*

`getDefaultSensor(SENSOR_TYPE_ACCELEROMETER)` *returns a non-wake-up sensor*

An accelerometer sensor reports the acceleration of the device along the 3 sensor axes. The measured acceleration includes both the physical acceleration (change of velocity) and the gravity. The measurement is reported in the x, y and z fields of sensors_event_t.acceleration.

All values are in SI units (m/s^2) and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Source: https://source.android.com/devices/sensors/sensor-types#accelerometer

`Sensor.TYPE_ACCELEROMETER`:

All values are in SI units (m/s^2)

- values[0]: Acceleration minus Gx on the x-axis
- values[1]: Acceleration minus Gy on the y-axis
- values[2]: Acceleration minus Gz on the z-axis

A sensor of this type measures the acceleration applied to the device (**Ad**). Conceptually, it does so by measuring forces applied to the sensor itself (**Fs**) using the relation:

Ad = - ∑Fs / mass

In particular, the force of gravity is always influencing the measured acceleration:

Ad = -g - ∑F / mass

For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of **g** = 9.81 m/s^2

Source: https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Claim 1

a six-axis motion sensor module attached to the PCB, comprising…an **accelerometer** for detecting and generating a **second signal set** comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the spatial pointer reference frame; and

Variable `a`, used by the `handleAcc()` function in the fusion.cpp file, represents acceleration data or a **second signal set**.

```
status_t Fusion::handleAcc(const vec3_t& a, float dT) {
    if (!checkInitComplete(ACC, a, dT))
        return BAD_VALUE;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

a six-axis motion sensor module attached to the PCB, comprising…an accelerometer for detecting and generating a second signal set comprising axial accelerations A_x, A_y, A_z associated with said movements and rotations of the 3D pointing device in the **spatial pointer reference frame**; and

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

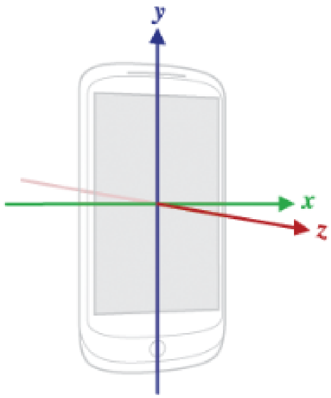


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves. This behavior is the same as the behavior of the OpenGL coordinate system.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

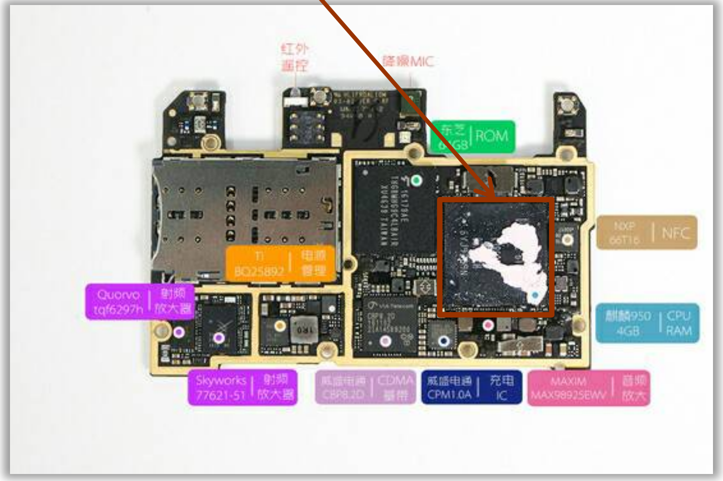
Claim 1

a processing and transmitting module, comprising a data transmitting unit electrically connected to the **six-axis motion sensor module** for transmitting said first and second signal sets thereof and a **computing processor** for receiving and calculating said first and second signal sets from the data transmitting unit,

computing processor

PLATFORM	OS	Android 6.0 (Marshmallow), upgradable to 7.0 (Nougat)
	Chipset	HiSilicon Kirin 950
	CPU	Octa-core (4x2.3 GHz Cortex-A72 & 4x1.8 GHz Cortex A53)
	GPU	Mali-T880 MP4
MEMORY	Card slot	microSD, up to 256 GB (uses SIM 2 slot)
	Internal	32/64 GB, 4 GB RAM
CAMERA	Primary	Dual 12 MP, f/2.2, 35mm, laser autofocus, dual-LED (dual tone) flash, check quality
	Features	1/2.9" sensor size, 1.25 µm pixel size, geo-tagging, touch focus, face detection, HDR, panorama
	Video	1080p@60fps, 1080p@30fps, 720p@120fps, check quality
	Secondary	8 MP, f/2.4, 1.4 µm pixel size
SOUND	Alert types	Vibration; MP3, WAV ringtones
	Loudspeaker	Yes
	3.5mm jack	Yes
		- Active noise cancellation with dedicated mic - DTS sound
COMMS	WLAN	Wi-Fi 802.11 a/b/g/n/ac, dual-band, WiFi Direct, hotspot
	Bluetooth	4.2, A2DP, EDR, LE
	GPS	Yes, with A-GPS, GLONASS/ BDS (market dependant)
	NFC	Yes
	Infrared port	Yes
	Radio	No
	USB	Type-C 1.0 reversible connector
FEATURES	Sensors	Fingerprint (rear-mounted), accelerometer, gyro, proximity, compass

Source: http://www.gsmarena.com/huawei_honor_8-8195.php



Source: https://www.androidheadlines.com/wp-content/uploads/2016/07/Honor-8-teardown-IT168_20.jpg

six-axis motion sensor module

Claim 1

communicating with the six-axis motion sensor module to calculate a **resulting deviation comprising resultant angles** in said spatial pointer reference frame by utilizing a comparison to compare the first signal set with the second signal set [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

Rotation vector

Underlying physical sensors: Accelerometer, Magnetometer, and Gyroscope

Reporting-mode: Continuous

`getDefaultSensor(SENSOR_TYPE_ROTATION_VECTOR)` *returns a non-wake-up sensor*

Source: https://source.android.com/devices/sensors/sensor-types#rotation_vector

getRotationMatrixFromVector

added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrixFromVector(float[],float[])

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[],float[])

Claim 1

communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by utilizing a comparison to compare the **first signal set** with the **second signal set** [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

The `predict()` function shows that the **first signal set** (angular velocities), `w`, is used to calculate the global variable `x0`.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
      x0 = O*q;
```

The **second signal set** (axial accelerations) `a`, is passed to the variable `z`, and used in the `update()` function to update the global variable `x0`.

```
      vec3_t unityA = a * l_inv;
```

```
      update(unityA, Ba, p);
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
```

```
    x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

communicating with the six-axis motion sensor module to calculate a resulting deviation comprising resultant angles in said spatial pointer reference frame by utilizing a **comparison to compare** the **first signal set** with the **second signal set** [Court's Construction: using the calculation of actual deviation angles to compare the first signal set with the second signal set] whereby said resultant angles in the spatial pointer reference frame of the resulting deviation of the six-axis motion sensor module of the 3D pointing device are obtained under said dynamic environments,

The `predict()` function and `update()` functions are used in sensor fusion to update the global variable `x0` in a quaternion form, which can represent actual deviation angles. In the `predict()` function, the **first signal set**, `w`, is used to calculate the global variable `x0`. In the `update()` function, `x0` is converted to the variable `Bb`. The **second signal set**, `a`, is passed to the `update()` function as local variable z, and is used by the `update()` function to update the global variable `x0`. The variable `Bb` (from the **first signal set**) and the variable `z` (from the **second signal set**) are **compared** to calculate the variable `e` on line 529 of the Fusion.cpp file. Therefore, during the calculation of actual deviation angles, the first signal set is compared with the second signal set.

```
430 void Fusion::predict(const vec3_t& w, float dT) {
431     const vec4_t q  = x0;
```

```
485     x0 = O*q;
```

```
495 void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
496     vec4_t q(x0);
497     // measured vector in body space: h(p) = A(p)*Bi
498     const mat33_t A(quatToMatrix(q));
499     const vec3_t Bb(A*Bi);
```

```
529     const vec3_t e(z - Bb);
```

```
533     x0 = normalize_quat(q);
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1
wherein the comparison utilized by the processing and transmitting module further comprises an update program to obtain an **updated state** based on a **previous state** associated with said **first signal set** and a **measured state** associated with said **second signal set**;
For example, the update program includes a `predict()` function and an `update()` function that are used to update the global variable `x0` based on `x0` (the **previous state**) associated with the **first signal set** `w` and `e` (the **measured state**) associated with the **second signal set** to calculate an **updated state** `x0`. The updated state `x0` becomes the previous state `x0` in the next iteration of the update program to obtain the updated state `x0` in that iteration.

first signal set

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
    x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

measured state

second signal set

```
    const vec3_t e(z - Bb);
    const vec3_t dq(K[0]*e);

    q += getF(q)*(0.5f*dq);
    x0 = normalize_quat(q);
```

next iteration

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 1

wherein the **measured state** includes a **measurement of said second signal set** and a **predicted measurement** obtained based on the first signal set **without using any derivatives of the first signal set**.

The variable e is a **measured state** that includes a **measurement of said second signal set** z and a **predicted measurement** Bb calculated based on x0 (the previous state, which is calculated based on the first signal set).

second signal set (measured accelerations)

```
    vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
```

measured state **second signal set** **predicted measurement**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the first signal set**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 4

The 3D pointing device of claim 1, wherein the spatial pointer reference frame is a reference frame in three dimensions; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

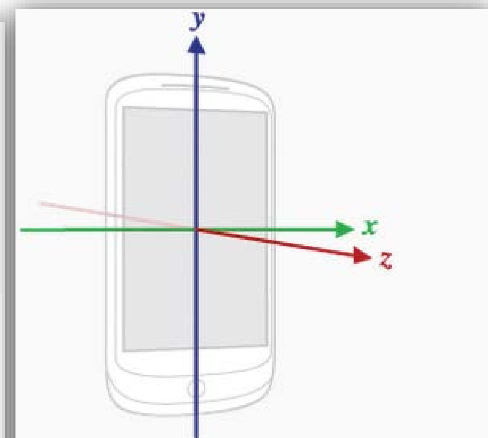
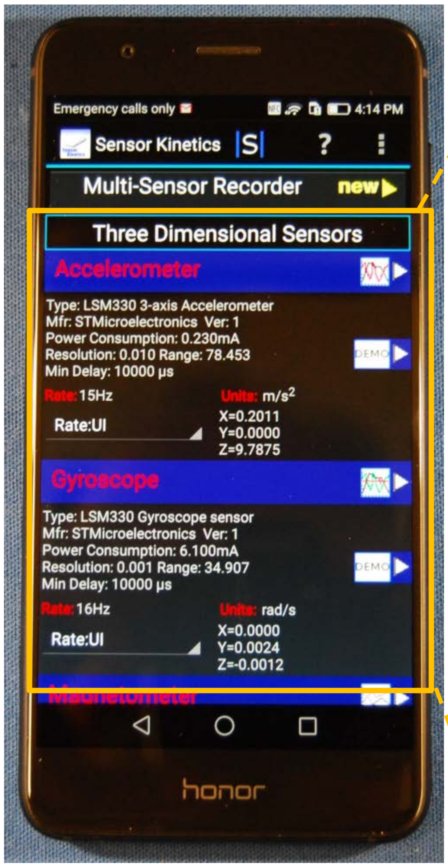


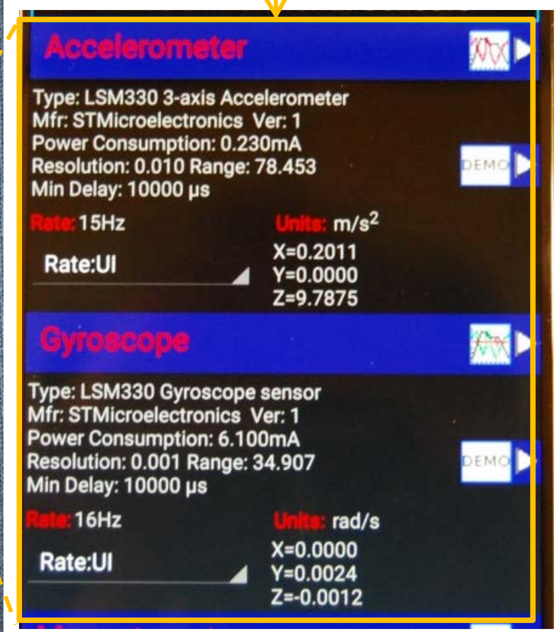
Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

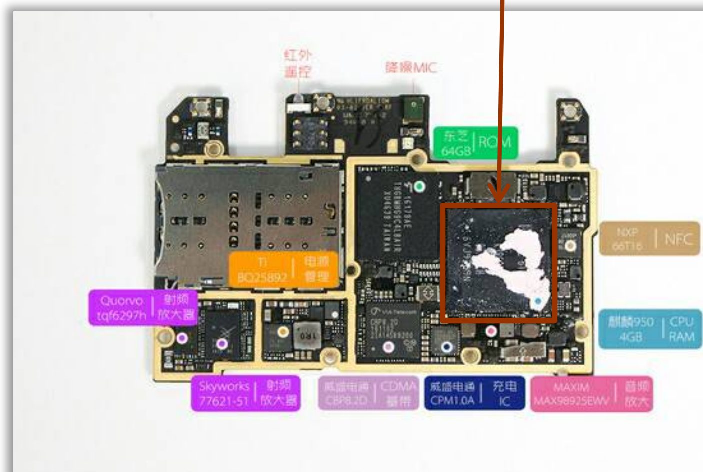
Claim 5
The 3D pointing device of claim 1, wherein the **data transmitting unit** of the processing and transmitting module is attached to the PCB enclosed by the housing and transmits said first and second signal of the **six-axis motion sensor module** to the **computer processor** via electronic connections.
The **computer processor** and the **six-axis motion sensor module** are each attached to the PCB, as is the **data transmitting unit**, which transmits the first and second signal of the **six-axis motion sensor module** to the **computer processor** via electronic connections.



six-axis motion sensor module



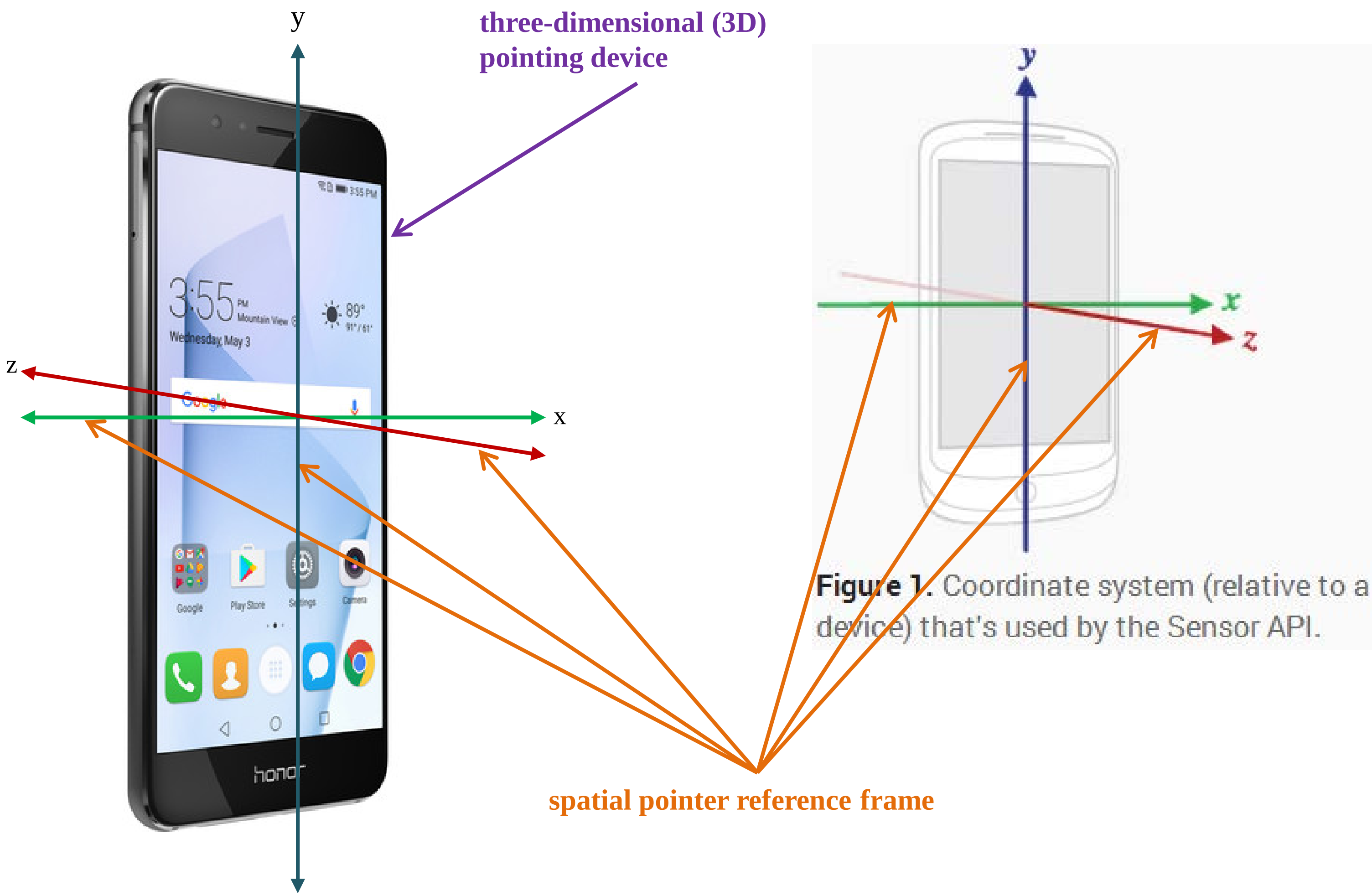
computing processor



Source: https://www.androidheadlines.com/wp-content/uploads/2016/07/Honor-8-teardown-IT168_20.jpg

Claim 14

A method for obtaining a resulting deviation including resultant angles in a spatial pointer reference frame of a three-dimensional (3D) pointing device utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said spatial pointer reference frame, comprising the steps of:



Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 14
obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable x0 based on x0 (the **previous state**) associated with **previous angular velocities** w gained at a previous time T-1 to obtain an updated state x0. The updated state x0 becomes the previous state x0 at time T (the next iteration) of the update program to obtain the updated state x0 at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
        x0 = O*q;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;

The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as `x0`. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, `w`, associated with the **current state**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
        x0 = O*q;
```

current state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
    vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
    const vec3_t e(z - Bb);
```

measured state **measured axial accelerations** **predicted axial accelerations**

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;

As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

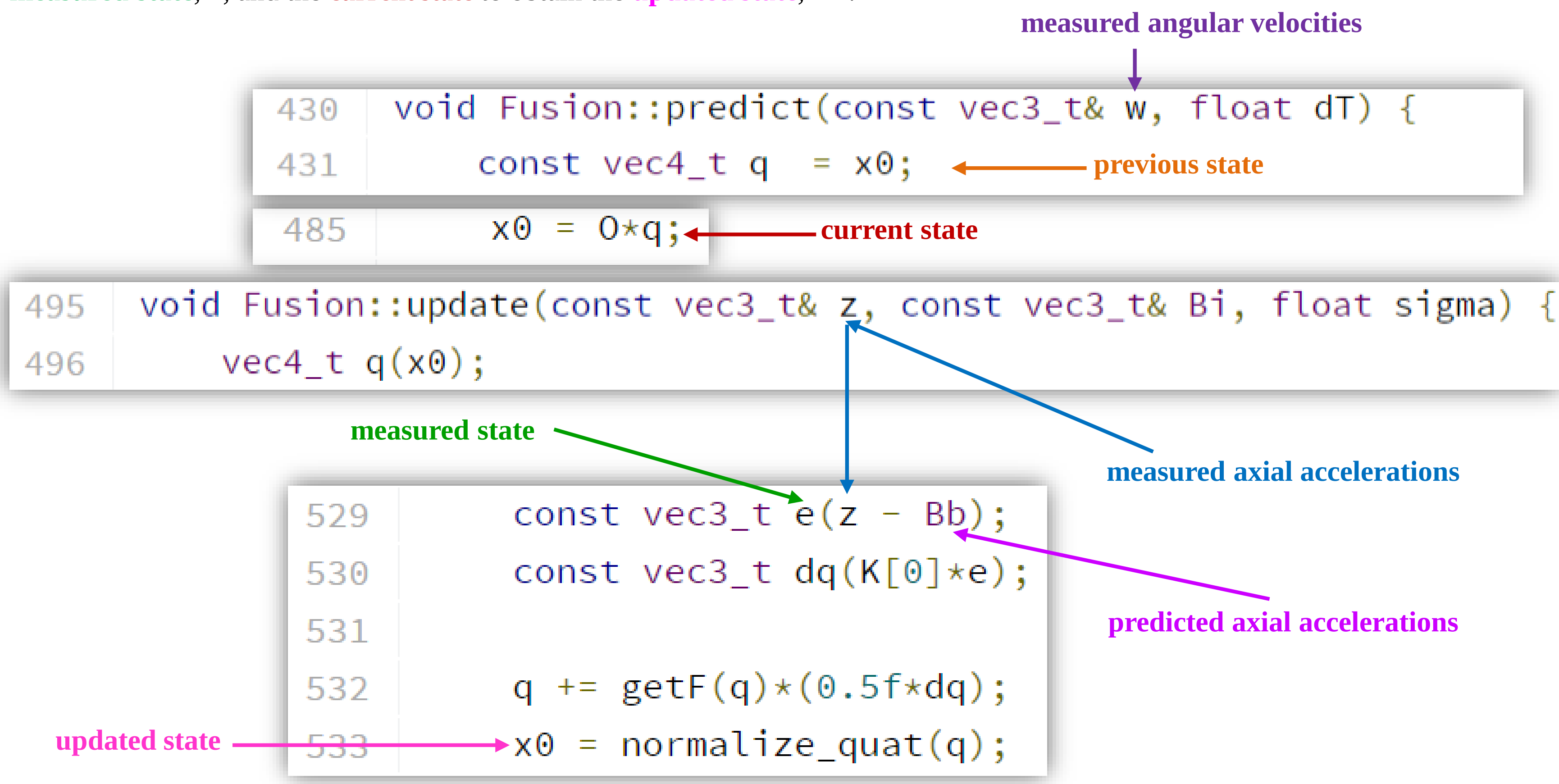
```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and

For example, as previously shown, the **measured state**, `e`, is obtained using the `update()` function, which combines the **measured axial accelerations**, `z`, and the **predicted axial accelerations**, `Bb`. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, `e`, and the **current state** to obtain the **updated state**, `x0`.



Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 14

calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.

The **updated state** `x0` is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, `x0`, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Claim 15

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the **updated state** of the six-axis motion sensor module to the **previous state** of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

For example, Android's source code discloses an iterative process for updating device motion. The **updated state** `x0` output at time T-1 becomes an input of the **previous state** at time T and the "state" is iteratively updated.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

updated state

```
x0 = normalize_quat(q);
```

next iteration

Moreover, the `getOrientation()` function outputs **yaw, pitch and roll angles**.

```
    public static float[] getOrientation(float[] R, float values[]) {
        if (R.length == 9) {
            values[0] = (float)Math.atan2(R[1], R[4]);
            values[1] = (float)Math.asin(-R[7]);
            values[2] = (float)Math.atan2(-R[6], R[8]);
        } else {
            values[0] = (float)Math.atan2(R[1], R[5]);
            values[1] = (float)Math.asin(-R[9]);
            values[2] = (float)Math.atan2(-R[8], R[10]);
        }
```

Source: https://android.googlesource.com/platform/frameworks/base/+/b267554/core/java/android/hardware/SensorManager.java

Claim 15

The method for obtaining a resulting deviation of a 3D pointing device of claim 14, further comprises the step of outputting the updated state of the six-axis motion sensor module to the previous state of the six-axis motion sensor module; and wherein said resultant angles of the resulting deviation includes **yaw, pitch and roll angles** about each of three orthogonal coordinate axes of the spatial pointer reference frame.

getOrientation

added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])

Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

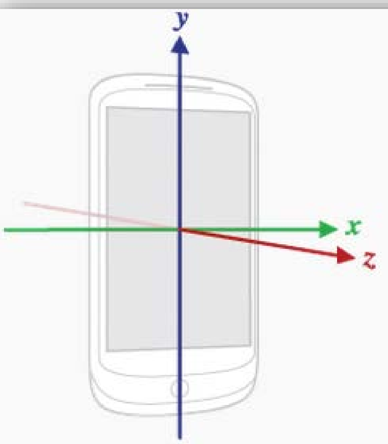


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 16
The method for obtaining a resulting deviation of a 3D pointing device of claim 14, wherein said **previous state** of the six-axis motion sensor module is a **first quaternion** with respect to said previous time T−1; and said **updated state** of the six-axis motion sensor module is a **third quaternion** with respect to said current time T.

The **previous state** set by the `predict()` function takes the form of a **first quaternion**, x0.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

The `update()` function calculates a **third quaternion** representing the **updated state**, x0.

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

updated state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 17

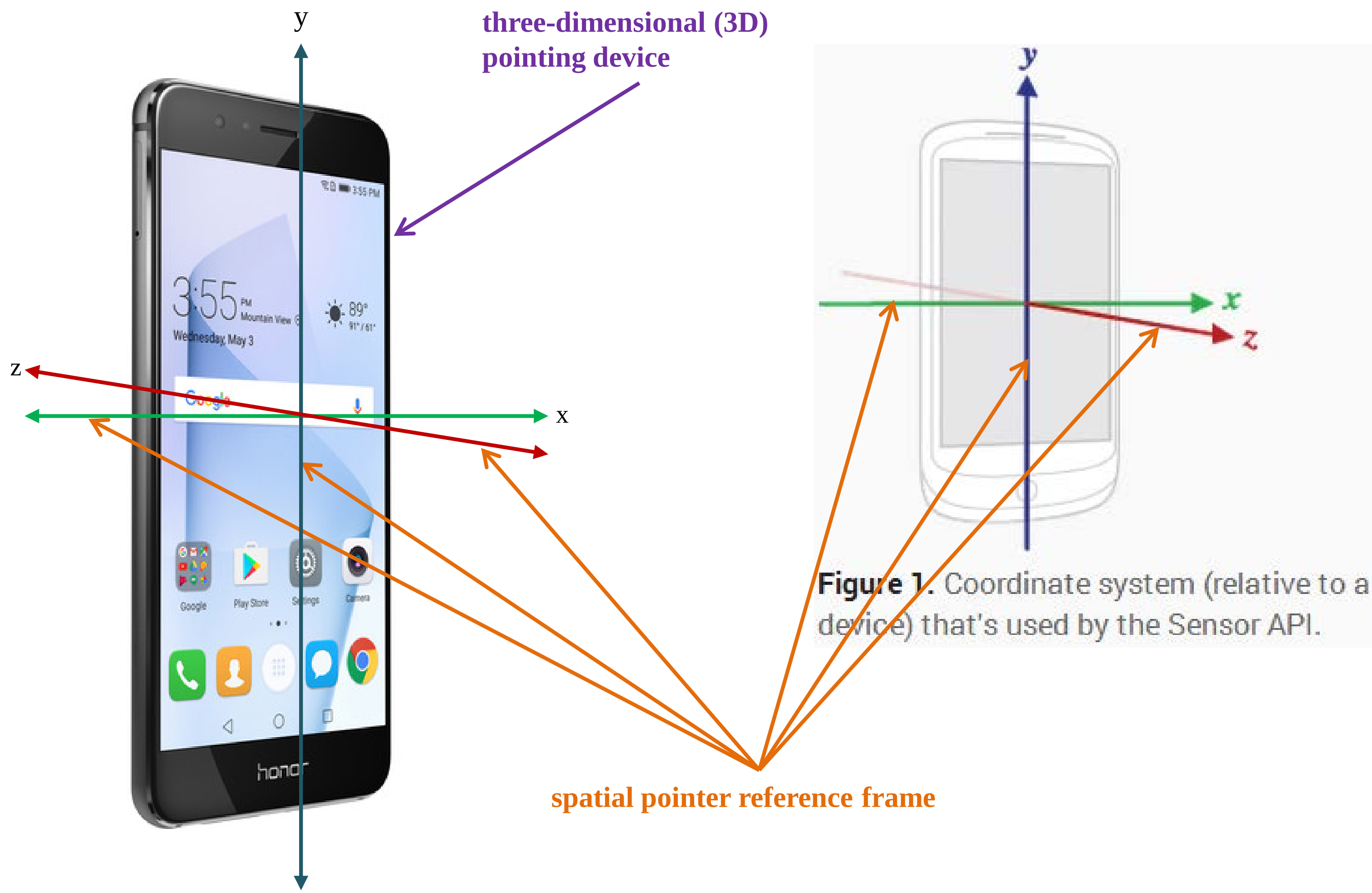
The method for obtaining a resulting deviation of 3D pointing device of claim 14, wherein the obtaining of said previous state of the six-axis motion sensor module further comprises initializing said **initial-value set**.
The fusion algorithm sets an **initial-value set** as shown in the `initFusion()` function.

```
void Fusion::initFusion(const vec4_t& q, float dT)
{
    // initial estimate: E{ x(t0) }
    x0 = q;
    x1 = 0;
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

A method for obtaining a resulting deviation including resultant angles in a spatial pointer reference frame of a three-dimensional (3D) pointing device utilizing a six-axis motion sensor module therein and subject to movements and rotations in dynamic environments in said spatial pointer reference frame, comprising the steps of:



Source: http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Claim 19

obtaining a **previous state** of the six-axis motion sensor module; wherein the **previous state** includes an initial-value set associated with **previous angular velocities** gained from the motion sensor signals of the six-axis motion sensor module at a previous time T−1;

The previous state is obtained through an update program that includes a `predict()` function and an `update()` function. Those functions that are used to update the global variable `x0` based on `x0` (the **previous state**) associated with **previous angular velocities** `w` gained at a previous time T-1 to obtain an updated state `x0`. The updated state `x0` becomes the previous state `x0` at time T (the next iteration) of the update program to obtain the updated state `x0` at time T.

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

previous state

```
        x0 = O*q;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
```

```
        const vec3_t e(z - Bb);
        const vec3_t dq(K[0]*e);

        q += getF(q)*(0.5f*dq);
        x0 = normalize_quat(q);
```

next iteration

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

obtaining a **current state** of the six-axis motion sensor module by obtaining **measured angular velocities** ω_x, ω_y, ω_z gained from the motion sensor signals of the six-axis motion sensor module at a current time T;

The `predict()` function runs during each iteration of the fusion algorithm, at a time T its output represents a **current state** output as `x0`. The `predict()` function is called by the `handleGyro()` function and receives **measured angular velocities**, `w`, associated with the **current state**.

```
void Fusion::handleGyro(const vec3_t& w, float dT) {
    if (!checkInitComplete(GYRO, w, dT))
        return;
```

measured angular velocities

```
void Fusion::predict(const vec3_t& w, float dT) {
    const vec4_t q  = x0;
```

```
        x0 = O*q;
```

current state

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

obtaining a **measured state** of the six-axis motion sensor module by obtaining **measured axial accelerations** Ax, Ay, Az gained from the motion sensor signals of the six-axis motion sensor module at the current time T and calculating **predicted axial accelerations** Ax′, Ay′, Az′ based on the **measured angular velocities** ω_x, ω_y, ω_z of the current state of the six-axis motion sensor module **without using any derivatives of the measured angular velocities** ωx, ωy, ωz;

The variable e is a **measured state** that includes **measured axil accelerations** z and **predicted axial accelerations** Bb calculated based on x0 (the previous state, which is calculated based on the **measured angular velocities**).

measured axial accelerations

```
        vec3_t unityA = a * l_inv;
```

```
void Fusion::update(const vec3_t& z, const vec3_t& Bi, float sigma) {
    vec4_t q(x0);
    // measured vector in body space: h(p) = A(p)*Bi
    const mat33_t A(quatToMatrix(q));
    const vec3_t Bb(A*Bi);
```

```
        const vec3_t e(z - Bb);
```

measured state

measured axial accelerations

predicted axial accelerations

As shown in the code above, the predicted measurement is obtained based on the first signal set **without using any derivatives of the measured angular velocities**.

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

said **current state** of the six-axis motion sensor module is a second quaternion with respect to said current time T;

As shown in the examples provided, the **current state** is represented by the global state variable x0, which is a quaternion with respect to the current time T.

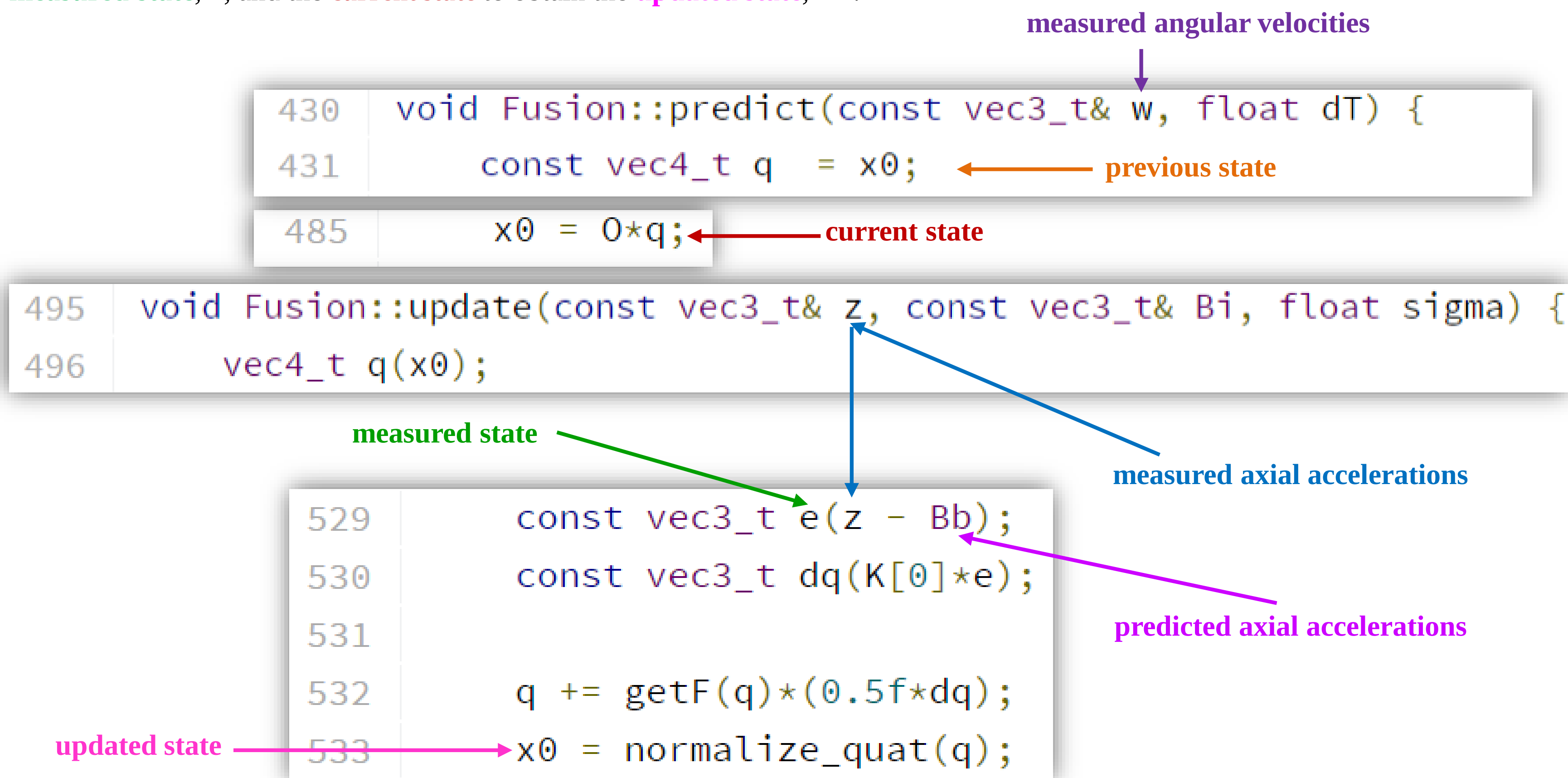
```
vec4_t Fusion::getAttitude() const {
    return x0;
}
```

Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19

comparing the second quaternion in relation to the **measured angular velocities** ω_x, ω_y, ω_z of the **current state** at current time T with the **measured axial accelerations** Ax, Ay, Az and the **predicted axial accelerations** Ax′, Ay′, Az′ also at current time T; obtaining an **updated state** of the six-axis motion sensor module by comparing the **current state** with the **measured state** of the six-axis motion sensor module; and

For example, as previously shown, the **measured state**, e, is obtained using the update() function, which combines the **measured axial accelerations**, z, and the **predicted axial accelerations**, Bb. Moreover, the **predicted axial accelerations** are determined based on the **measured angular velocities** of the **current state** at the current time T. The update() function further compares the **measured state**, e, and the **current state** to obtain the **updated state**, x0.



Source: https://android.googlesource.com/platform/frameworks/native/+/master/services/sensorservice/Fusion.cpp

Claim 19
calculating and converting the **updated state** of the six axis motion sensor module to said **resulting deviation comprising said resultant angles** in said spatial pointer reference frame of the 3D pointing device.

The **updated state** `x0` is in quaternion form, and can easily be converted to resultant angles.

According to Android's developer library, the `getOrientation()` function "computes the device's orientation based on the rotation matrix," and returns **resultant angles** including the Azimuth, Pitch, and Roll angles.

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

The `getRotationMatrixFromVector()` function "convert[s] a rotation vector to a rotation matrix," and the `getQuaternionFromVector()` function "convert[s] a rotation vector to a normalized quaternion." Therefore, the quaternion, `x0`, can be easily converted to its mathematically equivalent form, rotation matrix, and used by `getOrientation()` function to compute the orientation in its angular form.

Source: https://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])